Ministerul Educatiei, Culturii și Cercetarii al Republicii Moldova Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică şi Microelectronică
Departamentul Ingineria Software și Automatica

Raport
pentru lucrarea de laborator Nr. 1
la cursul „Metode criptografice de protecție a informației”

A efectuat:
Konjevic Alexandra, gr. FAF-213
A verificat: Aureliu ZGUREANU

Chișinău – 2023

Subject: Caesar Cipher

Objectives:

Task 1.1. Implement the Caesar algorithm for the English language alphabet in one of the programming languages. Use only the letter encoding as shown in Table 1 (using language-specific encodings such as ASCII or Unicode is not allowed). Key values will range from 1 to 25 inclusive, and no other values are allowed. The character values in the text are between 'A' and 'Z', 'a' and 'z', and no other values are permitted. If the user enters other values, they will be prompted with the correct range. Before encryption, the text will be converted to uppercase, and spaces will be removed. The user will be able to choose the operation - encryption or decryption, input the key, the message, or the ciphertext, and obtain the corresponding ciphertext or decrypted message.

Task 1.2. Implementing the Caesar algorithm with 2 keys, while preserving the conditions expressed in Task 1.1. Additionally, key 2 must contain only letters from the Latin alphabet and must have a length of at least 7.

Task 1.3. For this task, students will pair up. Each of them will encrypt a message consisting of 7-10 symbols (without spaces and written only in uppercase) using the Caesar cipher permutation version, with each person choosing their own keys. The resulting ciphertexts will be exchanged with their partner, along with the respective keys. Each of the two will perform the decryption and compare it with their partner's original version.

Theory:

The Caesar Cipher, also known as the Caesar Shift or Caesar's Code, is one of the simplest and earliest encryption techniques in the history of cryptography. Named after Julius Caesar, who is believed to have used it for secure communication during his military campaigns, this cipher represents a fundamental concept in the field of data security.

The essence of the Caesar Cipher lies in its straightforward approach to encryption. It operates by shifting each letter in the plaintext by a fixed number of positions down or up the alphabet. This shifting process, known as the "key" or "shift value," results in the generation of ciphertext, which conceals the original message from unintended readers.

- Encryption Formula. The Caesar Cipher's encryption process is remarkably simple. It involves shifting each letter in the plaintext message a fixed number of positions down or up the alphabet. This fixed number is known as the "key" or "shift value." The encryption formula can be expressed as:

$$E(x) = (x + k) \bmod 26$$

- In this formula:
  - E(x) represents the encrypted letter.
  - x represents the original letter's position in the alphabet (0 to 25, for 'A' to 'Z').
  - k represents the shift value, the number of positions each letter is moved.
  - The modulus operator (mod 26) ensures that the result remains within the bounds of the 26-letter English alphabet.

- Decryption Formula. Decrypting a Caesar Cipher message is essentially the reverse of the encryption process. To decrypt, you shift each letter in the ciphertext message backward by the same key value. The decryption formula is:

$$D(x) = (x - k) \bmod 26$$

  - In this formula:
    - D(x) represents the decrypted letter.
    - x represents the position of the ciphertext letter in the alphabet.
    - k is, again, the shift value.
    - The modulus operator ensures that the result remains within the alphabet range.

Task Completion:

I chose to write my program in JavaScript. First of all, in the `main()` function, the program reads the input from the user:

```
const operation = getOperation();
const numberOfKeys = getNumberOfKeys();
const text = deleteSpaces(getText().toUpperCase());
const key = getKey();
```

Fig 1. User Input

In case that the `numberOfKeys` value is 2, there is another prompt, to execute the encryption or decryption with the second key.

`getOperation()`, `getNumberOfKeys()`, `getText()`, `getKey()`, `getSecondKey()` are functions for validating the input of the user.

Also, in the `main()` function there is the string with all the characters from the alphabet:

```
var alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

Fig 2. Alphabet Declaration

Encryption:

The function `encrypt()` has three parameters: text (the text to encrypt), key and alphabet:

```
function encrypt(text, key, alphabet) {
    const indexes = getIndexes(text, alphabet);
    const encryptedIndexes = indexes.map((index) => (index + parseInt(key)) % 26);
    return encryptedIndexes.map((index) => alphabet[index]).join("");
}
```

Fig 3. Encryption Function

First of all, it uses the function `getIndexes()` to create an array with the indexes of all letters from the message to encrypt, that correspond to the indexes of the same letters in the `alphabet`:

```
function getIndexes(text, alphabet) {
    return text.split("").map((letter) => alphabet.indexOf(letter));
}
```

Fig 4. `getIndexes()` Function

After that, the function `encrypt()` creates an array of encrypted indexes, using the formula given above, in the theoretical part of the report, and creates a string with all the letters from the alphabet that correspond to the encrypted indexes.

Decryption:

This function works in a similar way with the `encrypt()`, but it uses the other formula to calculate decrypted indexes:

```
function decrypt(text, key, alphabet) {
    const indexes = getIndexes(text, alphabet);
    const decryptedIndexes = indexes.map((index) => {
        if (index !== -1) {
            let newIndex = (index - parseInt(key)) % 26;
            if (newIndex < 0) {
                newIndex = 26 + newIndex;
            }
            return newIndex;
        }
    });
    return decryptedIndexes
        .map((index) => (index !== undefined ? alphabet[index] : ""))
        .join("");
}
```

Fig 5. Decryption Function

It also has additional checks for the cases when the index is negative.

Second key case:

When the user chooses to add the second key to the cypher, `getSecondKey()` is executed.

```
if (numberOfKeys === "2") {
    const secondKey = getSecondKey().toUpperCase();
    alphabet = insertSecondKey(secondKey, alphabet).join("");
}
```

Fig 6. Second Key For Encryption/Decryption

After that, the original `alphabet` is replaced with the one returned by the function `insertSecondKey()` - an alphabet that has at the begging the second key.

```
function insertSecondKey(secondKey, alphabet) {
    var alphabetWithSecondKey = (secondKey + alphabet).split("");
    return [...new Set(alphabetWithSecondKey)];
}
```

Fig 7. `insertSecondKey()` Function

The alphabet is converted to a set, to eliminate the duplicates, and after that, with the spread operator, it is converted once again to an array.

After that, the new alphabet can be used in the `encrypt()` and `decrypt()` functions.

Outputs:

Input validation examples:

```
Enter operation (e/d): dfg
Invalid input. Please try again.
Enter operation (e/d): r
Invalid input. Please try again.
Enter operation (e/d): e
Enter number of keys (1/2): 30
Invalid input. Please try again.
Enter number of keys (1/2): 1
Enter text: some_text!
Invalid input. Please try again.
Enter text: some_text123123
Invalid input. Please try again.
Enter text: sometext
Enter key: 100
Invalid input. Please try again.
Enter key: -12
Invalid input. Please try again.
Enter key: 3
VRPHWHAW
```

Fig 8. Output With Different Validation

1.1:

```
Enter operation (e/d): e
Enter number of keys (1/2): 1
Enter text: cifrul cezar
Enter key: 3
FLIUXOFHCDU
```

Fig 9.a. Encryption With One Key

```
Enter operation (e/d): d
Enter number of keys (1/2): 1
Enter text: FLIUXOFHCDU
Enter key: 3
CIFRULCEZAR
```

Fig 9.b. Decryption With One Key

1.2:

```
Enter operation (e/d): e
Enter number of keys (1/2): 2
Enter text: cifrul cezar
Enter key: 10
Enter second key: somerandomkey
UZVHAOUGYIH
```

Fig 10.a. Encryption With Two Keys

```
Enter operation (e/d): d
Enter number of keys (1/2): 2
Enter text: UZVHAOUGYIH
Enter key: 10
Enter second key: somerandomkey
CIFRULCEZAR
```

Fig 10.b. Decryption With Two Keys

1.3:

```
Enter operation (e/d): d
Enter number of keys (1/2): 2
Enter text: JAZGXIEG
Enter key: 20
Enter second key: loremipsum
WHITECAT
```

Fig 11. Decrypted Message Received From Colleague

Conclusion:

As I conclude the exploration of the Caesar Cipher, it is crucial to recognize its historical significance, its educational value, and its enduring relevance as a cornerstone of cryptographic knowledge. It remains a testament to the enduring quest for secure communication and data protection, laying the groundwork for the intricate cryptographic systems that safeguard our digital world.

In closing, the Caesar Cipher, with its timeless charm and enduring lessons, serves as a bridge that connects the past to the present, reminding us of the ever-evolving nature of data security and encryption. It beckons us to explore further, to innovate, and to safeguard the confidentiality of our communications in an increasingly interconnected world.

GitHub Repository: https://github.com/alya1007/Labs-semester-5/tree/master/CS